DS practical guys mt kr laadleee

Practical. P1(A):- DECLARE ARRAY AND DISPLAY ARRAY

```
#include <stdio.h>

int main() {

   int numbers[5] = {10, 20, 30, 40, 50};

   printf("Array elements are:\n");
   for (int i = 0; i < 5; i++) {
      printf("numbers[%d] = %d\n", i, numbers[i]);
   }
   return 0;
}
```

Prectical.P1(B):-INSERT ELEMENT, DELETE.

```
#include <stdio.h>

int main() {
   int arr[100], n, i, pos, value;

   printf("Enter number of elements: ");
   scanf("%d", &n);

   printf("Enter %d elements: ", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   printf("\nEnter position to insert (1 to %d): ", n + 1);
   scanf("%d", &pos);
   printf("Enter value to insert: ");
   scanf("%d", &value);

   if (pos < 1 || pos > n + 1) {
      printf("Invalid position!\n");
   } else {

      for (i = n; i >= pos; i--) {
```

```
            arr[i] = arr[i - 1];
        }
        arr[pos - 1] = value;
        n++;
    }

    printf("Array after insertion: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("\nEnter position to delete (1 to %d): ", n);
    scanf("%d", &pos);

    if (pos < 1 || pos > n) {
        printf("Invalid position!\n");
    } else {

        for (i = pos - 1; i < n - 1; i++) {
            arr[i] = arr[i + 1];
        }
        n--;
    }

    printf("Array after deletion: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

P2(A):-Write a program to implement the concept of stack with push-pop display operation in c

```
#include <stdio.h>
#define MAX 5

int stack[MAX];
```

```
int top = -1;

void push() {
    int value;
    if (top == MAX - 1) {
        printf("Stack Overflow! Cannot push element.\n");
    } else {
        printf("Enter value to push: ");
        scanf("%d", &value);
        top++;
        stack[top] = value;
        printf("%d pushed into stack.\n", value);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow! Cannot pop element.\n");
    } else {
        printf("%d popped from stack.\n", stack[top]);
        top--;
    }
}

void display() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice;
```

```
    do {
        printf("\n***** STACK MENU *****\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

P2(B):-WRITE A PROGRAM TO COVERT AN INFIX EXPRESSION TO POSTFIX CONVERSION

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

char stack[30], infix[30], postfix[30];
int top = -1;
```

```
int ifOperand(char a) {
   if ((a >= '0' && a <= '9') || (a >= 'a' && a <= 'z') || (a >= 'A' && a <= 'Z')) {
      return 1;
   } else {
      return 0;
   }
}

int ifOperator(char a) {
   if (a == '%' || a == '*' || a == '/' || a == '+' || a == '-') {
      return 1;
   } else {
      return 0;
   }
}

int push(char a) {
   if (top >= 29) {
      return 1;
   } else {
      top++;
      stack[top] = a;
      stack[top + 1] = '\0';
      return 0;
   }
}

int pop() {
   char a;
   if (top == -1) {
      return '1';
   } else {
      a = stack[top];
      stack[top] = 0;
      top--;
      return a;
   }
```

```
}

int priority(char a) {
   if (a == '%') {
      return 3;
   } else if (a == '*' || a == '/') {
      return 2;
   } else if (a == '+' || a == '-') {
      return 1;
   } else {
      return 0;
   }
}

int main() {
   int i, j;
   char a;

   printf("Enter the infix expression:\n");
   scanf("%s", infix);

   printf("Postfix expression after every iteration:\n");

   for (i = 0, j = 0; i < strlen(infix); i++) {
      if (ifOperand(infix[i])) {
         postfix[j] = infix[i];
         j++;
      }
      else if (ifOperator(infix[i])) {
         while (strlen(stack) && priority(infix[i]) <= priority(stack[top])) {
            a = pop();
            if (a == '1') {
               printf("The infix is wrong\n");
               break;
            }
            postfix[j] = a;
            j++;
         }
```

```
            push(infix[i]);
        }
        else if (infix[i] == ')') {
            a = pop();
            while (a != '(') {
                postfix[j] = a;
                j++;
                a = pop();
            }
        }
        else if (infix[i] == '(') {
            push(infix[i]);
        }

        printf("%s\n", postfix);
    }

    while (top > -1) {
        postfix[j] = pop();
        j++;
    }

    postfix[j] = '\0';
    printf("The postfix expression is: %s\n", postfix);

    return 0;
}
```

P2(C):-WRITE A PROGRAM TO IMPLEMENT TOWER OF HANOI PROBLEM

```
#include <stdio.h>

void move(int n, char source, char dest, char spare);

int main() {
    int n;
    printf("Enter the number of rings: ");
    scanf("%d", &n);
    move(n, 'A', 'C', 'B');
    return 0;
```

```
}

void move(int n, char source, char dest, char spare) {
   if (n == 1) {
      printf("Move from %c to %c\n", source, dest);
   } else {
      move(n - 1, source, spare, dest);
      move(1, source, dest, spare);
      move(n - 1, spare, dest, source);
   }
}
```

P3(A):-Write program to create link list and display node element

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node *next;
};

struct Node *head = NULL;

void createList(int n) {
   struct Node *newNode, *temp;
   int data, i;

   if (n <= 0) {
      printf("Number of nodes must be greater than 0.\n");
      return;
   }

   head = (struct Node *)malloc(sizeof(struct Node));
   if (head == NULL) {
      printf("Unable to allocate memory.\n");
      return;
   }

   printf("Enter data for node 1: ");
```

```
    scanf("%d", &data);
    head->data = data;
    head->next = NULL;
    temp = head;

    for (i = 2; i <= n; i++) {
        newNode = (struct Node *)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Unable to allocate memory.\n");
            break;
        }
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        temp->next = newNode;
        temp = newNode;
    }
}

void displayList() {
    struct Node *temp;
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    temp = head;
    printf("Linked List elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int n;
```

```
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    createList(n);
    displayList();

    return 0;
}
```

P3(B):-Write a program to create double linked list and - short the element in the linked list in C.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node *prev;
    struct Node *next;
};

struct Node *head = NULL;

void createList(int n) {
    struct Node *newNode, *temp;
    int data, i;

    if (n <= 0) {
        printf("Number of nodes must be greater than 0.\n");
        return;
    }

    head = (struct Node *)malloc(sizeof(struct Node));
    if (head == NULL) {
        printf("Unable to allocate memory.\n");
        return;
    }

    printf("Enter data for node 1: ");
    scanf("%d", &data);
```

```
    head->data = data;
    head->prev = NULL;
    head->next = NULL;
    temp = head;

    for (i = 2; i <= n; i++) {
        newNode = (struct Node *)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Unable to allocate memory.\n");
            break;
        }
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->prev = temp;
        newNode->next = NULL;
        temp->next = newNode;
        temp = newNode;
    }
}

void sortList() {
    struct Node *i, *j;
    int temp;

    if (head == NULL) {
        return;
    }

    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
```

```
}

void displayList() {
   struct Node *temp;
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }

   temp = head;
   printf("Doubly Linked List elements: ");
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

int main() {
   int n;

   printf("Enter number of nodes: ");
   scanf("%d", &n);

   createList(n);

   printf("\nOriginal List:\n");
   displayList();

   sortList();

   printf("\nSorted List:\n");
   displayList();

   return 0;
}
```

P4(A):-Write a program to declare queue and display queue in C.

```
#include <stdio.h>
```

```
#define SIZE 5

int queue[SIZE];
int front = -1, rear = -1;

void enqueue(int value) {
    if (rear == SIZE - 1) {
        printf("Queue is full! Cannot insert %d\n", value);
    } else {
        if (front == -1) front = 0;
        rear++;
        queue[rear] = value;
    }
}

void displayQueue() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {

    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);

    displayQueue();

    return 0;
}
```

P4(B):-WRITE A PROGRAM TO IMPLEMENT THE CONCEPT OF QUEUE WITH

INSERT,DELETE,DISPLAY,AND EXIT OPERATION

```
#include <stdio.h>

#define SIZE 6


int queue[SIZE];

int front = -1, rear = -1;


void insert() {

   int value;

   if (rear == SIZE - 1) {

      printf("Queue is full!\n");

   } else {

      printf("Enter the value to insert: ");

      scanf("%d", &value);

      if (front == -1) front = 0;

      rear++;

      queue[rear] = value;

      printf("%d inserted into the queue.\n", value);

   }

}


void deleteq() {

   if (front == -1 || front > rear) {

      printf("Queue is empty!\n");

   } else {

      printf("%d deleted from the queue.\n", queue[front]);

      front++;

   }

}


void display() {

   if (front == -1 || front > rear) {

      printf("Queue is empty!\n");

   } else {

      printf("Queue elements: ");

      for (int i = front; i <= rear; i++) {

         printf("%d ", queue[i]);
```

```
        }
        printf("\n");
    }
}

int main() {
    int choice;
    while (1) {
        printf("\n--- Queue Menu ---\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: insert(); break;
            case 2: deleteq(); break;
            case 3: display(); break;
            case 4: return 0;
            default: printf("Invalid choice! Try again.\n");
        }
    }
}
```

P5(a):- WRITE A PROGRAM TO IMPLEMENT BUBBLE SORT

```
#include <stdio.h>

int arr[20];


void bubbleSort(int n) {
    int i, j, temp;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                m
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
```

```
        }
    }
}

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Enter %dth element: ", i);
        scanf("%d", &arr[i]);
    }

    bubbleSort(n);

    printf("\nSorted elements:\n");
    for (i = 0; i < n; i++) {
        printf("%d\n", arr[i]);
    }

    return 0;
}
```

P5(B):- WRITE A PROGRAM TO IMPLEMENT INSERATION SORT

```
#include <stdio.h>
#include <malloc.h>

int a[20];

void insertion_sort(int n)
{
    int val, vacant, i;
    for (i = 1; i < n; i++)
    {
        val = a[i];
        vacant = i;

        while (a[vacant - 1] > val && vacant != 0)
```

```
    {
      a[vacant] = a[vacant - 1];
      vacant = vacant - 1;
    }
    a[vacant] = val;
  }
  printf("Successfully sorted using Insertion Sort Algorithm\n");
}

void main()
{
  int n, i;

  printf("Enter number of elements: ");
  scanf("%d", &n);

  for (i = 0; i < n; i++)
  {
    printf("Enter %dth element: ", i);
    scanf("%d", &a[i]);
  }

  insertion_sort(n);

  for (i = 0; i < n; i++)
  {
    printf("%d\n", a[i]);
  }
}
```

P6(A):- WRITE A PREPARE TO IMPLEMENT MERGE SORT

```
#include <stdio.h>

int arr[20];

void merge(int beg, int mid, int end) {
  int i = beg, j = mid + 1, index = beg, temp[20], k;

  while ((i <= mid) && (j <= end)) {
```

```
      if (arr[i] < arr[j]) {
         temp[index] = arr[i];
         i++;
      } else {
         temp[index] = arr[j];
         j++;
      }
      index++;
   }

   while (i <= mid) {
      temp[index] = arr[i];
      i++;
      index++;
   }

   while (j <= end) {
      temp[index] = arr[j];
      j++;
      index++;
   }

   for (k = beg; k < index; k++) {
      arr[k] = temp[k];
   }
}

void mergeSort(int beg, int end) {
   if (beg < end) {
      int mid = (beg + end) / 2;
      mergeSort(beg, mid);
      mergeSort(mid + 1, end);
      merge(beg, mid, end);
   }
}

int main() {
   int n, i;
```

```
   printf("Enter number of elements: ");
   scanf("%d", &n);

   for (i = 0; i < n; i++) {
      printf("Enter %dth element: ", i);
      scanf("%d", &arr[i]);
   }

   mergeSort(0, n - 1);

   printf("\nSorted Array:\n");
   for (i = 0; i < n; i++) {
      printf("%d\n", arr[i]);
   }

   return 0;
}
```

P6 (B)WRITE A PROGRAM TO SEARCH ELEMENT USING BINARY SEARCH

```
#include <stdio.h>
#include <string.h>
#define SIZE 30

int a[SIZE], n;
int binary(int a[], int val);

int main() {
   int val = 0, pos, i;
   char op = 'y';

   printf("Enter the number of elements: ");
   scanf("%d", &n);

   printf("Enter elements in sorted order:\n");
   for(i = 0; i < n; i++) {
      printf("Enter a value: ");
      scanf("%d", &a[i]);
   }
```

```
   while(op != 'n') {
      printf("\nEnter value to be searched: ");
      scanf("%d", &val);

      pos = binary(a, val);
      if(pos == -1)
         printf("Value not found\n");
      else
         printf("Value found at position %d\n", (pos + 1));

      printf("Do you want to continue (y/n): ");
      scanf(" %c", &op);
   }

   return 0;
}

int binary(int a[], int val) {
   int beg, end, mid;
   beg = 0;
   end = n - 1;

   while(beg <= end) {
      mid = (beg + end) / 2;

      if(val == a[mid])
         return mid;
      else if(val < a[mid])
         end = mid - 1;
      else
         beg = mid + 1;
   }
   return -1;
}
```

P7(A)WRITE A PROGRAM TO CREATE THE TREE AND DISPLAY ELEMENT

P7(B)WRITE A PROGRAM TO CONSTRUCT THE BINARY TREE

```
#include <stdio.h>
```

```
#include <malloc.h>

struct node {
   int data;
   struct node *left;
   struct node *right;
};

struct node *root = NULL;

// Function prototypes
struct node *create(struct node *);
struct node *display(struct node *);
void preorder(struct node *temp);
void inorder(struct node *temp);
void postorder(struct node *temp);

int main() {
   int choice;
   do {
      printf("\n***** MAIN MENU *****\n");
      printf("1. Create a binary search tree\n");
      printf("2. Display the tree\n");
      printf("3. EXIT\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      printf("\n");

      switch (choice) {
         case 1: root = create(root);
               break;
         case 2: root = display(root);
               break;
         case 3: break;
      }
   } while (choice != 3);
   return 0;
}
```

```
// Function to create BST
struct node *create(struct node *root) {
   struct node *newnode = NULL, *temp = NULL, *parent = NULL;
   int val;

   printf("Enter the data or enter -1 to exit: ");
   scanf("%d", &val);

   while (val != -1) {
      newnode = (struct node *)malloc(sizeof(struct node));
      newnode->data = val;
      newnode->left = NULL;
      newnode->right = NULL;

      if (root == NULL) {
         root = newnode;
      } else {
         temp = root;
         while (temp != NULL) {
            parent = temp;
            if (val < temp->data)
               temp = temp->left;
            else
               temp = temp->right;
         }
         if (val < parent->data)
            parent->left = newnode;
         else
            parent->right = newnode;
      }
      printf("Enter the data or enter -1 to exit: ");
      scanf("%d", &val);
   }

   printf("Successfully created\n");
   return root;
}
```

```
// Function to display traversals
struct node *display(struct node *root) {
   int choice1;
   printf("\n----- DISPLAY MENU -----\n");
   printf("1. Pre-Order\n");
   printf("2. In-Order\n");
   printf("3. Post-Order\n");
   printf("4. EXIT\n");
   printf("Enter your choice: ");
   scanf("%d", &choice1);

   switch (choice1) {
      case 1: printf("\nThe Pre-Order Traversal is: ");
            preorder(root);
            break;
      case 2: printf("\nThe In-Order Traversal is: ");
            inorder(root);
            break;
      case 3: printf("\nThe Post-Order Traversal is: ");
            postorder(root);
            break;
      case 4: break;
   }
   printf("\n");
   return root;
}

// Traversal functions
void preorder(struct node *temp) {
   if (temp != NULL) {
      printf("%d ", temp->data);
      preorder(temp->left);
      preorder(temp->right);
   }
}

void inorder(struct node *temp) {
```

```
  if (temp != NULL) {
    inorder(temp->left);
    printf("%d ", temp->data);
    inorder(temp->right);
  }
}

void postorder(struct node *temp) {
  if (temp != NULL) {
    postorder(temp->left);
    postorder(temp->right);
    printf("%d ", temp->data);
  }
}
```

P8(A):-write a program to generate a as join matrix

```
#include <stdio.h>

#define MAX 10

// Function to get cofactor of A[p][q]
void getCofactor(int A[MAX][MAX], int temp[MAX][MAX], int n, int p, int q) {
  int i = 0, j = 0;
  for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
      // Skip current row and column
      if (row != p && col != q) {
        temp[i][j++] = A[row][col];

        // Row is filled, move to next row
        if (j == n - 1) {
          j = 0;
          i++;
        }
      }
    }
  }
}
```

```
// Function to calculate determinant of matrix
int determinant(int A[MAX][MAX], int n) {
   if (n == 1)
      return A[0][0];

   int temp[MAX][MAX];
   int sign = 1, det = 0;

   for (int f = 0; f < n; f++) {
      getCofactor(A, temp, n, 0, f);
      det += sign * A[0][f] * determinant(temp, n - 1);
      sign = -sign; // alternate sign
   }
   return det;
}

// Function to calculate adjoint matrix
void adjoint(int A[MAX][MAX], int adj[MAX][MAX], int n) {
   if (n == 1) {
      adj[0][0] = 1;
      return;
   }

   int temp[MAX][MAX];
   int sign;

   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         // Get cofactor of A[i][j]
         getCofactor(A, temp, n, i, j);

         // sign is + if (i+j) is even, else -
         sign = ((i + j) % 2 == 0) ? 1 : -1;

         // adjoint is transpose of cofactor matrix
         adj[j][i] = sign * determinant(temp, n - 1);
      }
   }
```

```
}

// Function to print matrix
void printMatrix(int A[MAX][MAX], int n) {
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
         printf("%5d ", A[i][j]);
      printf("\n");
   }
}

int main() {
   int n;
   int A[MAX][MAX], adj[MAX][MAX];

   printf("Enter the order of matrix (n x n): ");
   scanf("%d", &n);

   printf("Enter the elements of matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &A[i][j]);

   printf("\nInput Matrix:\n");
   printMatrix(A, n);

   adjoint(A, adj, n);

   printf("\nAdjoint Matrix:\n");
   printMatrix(adj, n);

   return 0;
}
```

P8(b):-write an program to insert an element into maximum heap

```
/*
   Program Title: Binary Search Tree Traversals in C
   Description  : This program creates a Binary Search Tree (BST)
                  by inserting user-given values and then displays
```

```
        the tree using Inorder, Preorder, and Postorder traversals.
*/

#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node *left, *right;
};

// Create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Insert into Binary Search Tree
struct Node* insert(struct Node* root, int value) {
    if (root == NULL)
        return createNode(value);

    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);

    return root;
}

// Inorder Traversal (Left -> Root -> Right)
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
```

```
        inorder(root->right);
    }
}

// Preorder Traversal (Root -> Left -> Right)
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

// Postorder Traversal (Left -> Right -> Root)
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

// Main function
int main() {
    struct Node* root = NULL;
    int n, value;

    printf("Enter number of nodes: ");
    scanf("%d", &n);

    printf("Enter %d values:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("\nInorder Traversal  : ");
    inorder(root);
```

```
    printf("\nPreorder Traversal : ");
    preorder(root);

    printf("\nPostorder Traversal: ");
    postorder(root);

    return 0;
}
```